Postel

Network Working Group
Request For Comment: 1022

C. Partridge
BBN/NNSC
G. Trewitt
Stanford
October 1987

# THE HIGH-LEVEL ENTITY MANAGEMENT PROTOCOL (HEMP)

## STATUS OF THIS MEMO

An application protocol for managing network entities such as hosts, gateways and front-end machines, is presented. This protocol is a component of the High-Level Entity Management System (HEMS) described in RFC-1021. Readers may want to consult RFC-1021 when reading this memo. This memo also assumes a knowledge of the ISO data encoding standard, ASN.1. Distribution of this memo is unlimited.

## PROTOCOL OVERVIEW

The High-Level Entity Management Protocol (HEMP) provides an encapsulation system and set of services for communications between applications and managed entities. HEMP is an application protocol which relies on existing transport protocols to deliver HEMP messages to their destination(s).

The protocol is targeted for management interactions between applications and entities. The protocol is believed to be suitable for both monitoring and control interactions.

HEMP provides what the authors believe are the three essential features of a management protocol: (1) a standard encapsulation scheme for all interactions, (2) an authentication facility which can be used both to verify messages and limit access to managed systems, and (3) the ability to encrypt messages to protect sensitive information. These features are discussed in detail in the following sections.

## PROTOCOL OPERATION

HEMP is designed to support messages; where a message is an arbitrarily long sequence of octets.

Five types of messages are currently defined: request, event, reply, and protocol error, and application error messages. Reply, protocol error and application error messages are only sent in reaction to a request message, and are referred to collectively as responses.

Two types of interaction are envisioned: a message exchange between an application and an entity managed by the application, and unsolicited messages from an entity to the management centers responsible for managing it.

When an application wants to retrieve information from an entity or gives instructions to an entity, it sends a request message to the entity. The entity replies with a response, either a reply message if the request was valid, or an error message if the request was invalid (e.g., failed authentication). It is expected that there will only be one response to a request message, although the protocol does not preclude multiple responses to a single request.

Protocol error messages are generated if errors are found when processing the HEMP encapsulation of the message. The possible protocol error messages are described later in this document. Non-HEMP errors (e.g., errors that occur during the processing of the contents of the message) are application errors. The existence of application error messages does not preclude the possibility that a reply will have an error message in it. It is expected that the processing agent on the entity may have already started sending a reply message before an error in a request message is discovered. As a result, application errors found during processing may show up in the reply message instead of a separate application error message.

Note that in certain situations, such as on secure networks, returning error messages may be considered undesirable. As a result, entities are not required to send error messages, although on "friendly" networks the use of error messages is encouraged.

Event messages are unsolicited notices sent by an entity to an address, which is expected to correspond to one or more management centers. (Note that a single address may correspond to a multicast address, and thus reach multiple hosts.) Event messages are typically used to allow entities to alert management centers of important changes in their state (for example, when an interface goes down or the entity runs out of network buffers).

## STANDARD MESSAGE FORMAT

Every HEMP message is put in the general form shown in Figure 1.

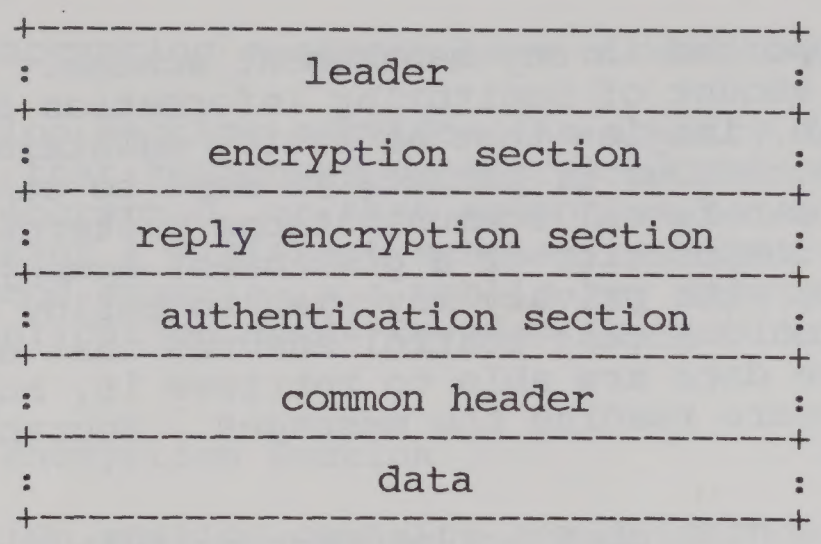


Figure 1: General Form of HEMP Messages

Each message has five components: (1) the leader, which is simply the ASN.1 tag and message length; (2) the encryption section, which provides whatever information the receiver may require to decrypt the message; (3) the reply encryption section, in which the requesting application may specify the type of encryption to use in the reply; (4) the authentication section, which allows the receiver to authenticate the message; (5) the common header, which identifies the message type, the HEMP version, and the message id; and (6) the data section. All four sections following the leader are also ASN.1 encoded. The ASN.1 format of the message is shown in Figure 2.

```
HempMessage ::= [0] IMPLICIT SEQUENCE {
    [0] IMPLICIT EncryptSection OPTIONAL,
    [1] IMPLICIT ReplyEncryptSection OPTIONAL,
    [2] IMPLICIT AuthenticateSection OPTIONAL,
    [3] IMPLICIT CommonHeader,
    [4] IMPLICIT Data }
```

Figure 2: ASN.1 Format of HEMP Messages

The ordering of the sections is significant. The encryption section comes first so that all succeeding sections (which may contain sensitive information) may be encrypted. The authentication section precedes the header so that messages which fail authentication can be discarded without header processing.

## THE ENCRYPTION SECTION

### Need For Encryption

Encryption must be supported in any management scheme. In particular, a certain amount of monitoring information is potentially sensitive. For example, imagine that an entity maintains a traffic matrix, which shows the number of packets it sent to other entities. Such a traffic matrix can reveal communications patterns in an organization (e.g., a corporation or a government agency). Organizations concerned with privacy may wish to employ encryption to protect such information. Access control ensures that only people entitled to request the data are able to retrieve it, but does not protect from eavesdroppers reading the messages. Encryption protects against eavesdropping.

Note that encryption in HEMP does not protect against traffic analysis. It is expected that HEMP interactions will have distinct signatures such that a party which can observe traffic patterns may guess at the sort of interactions being performed, even if the data being sent is encrypted. Organizations concerned with security at this level should additionally consider link-level encryption.

### Format of the Encryption Section

The encryption section contains any data required to decrypt the message. The ASN.1 format of this section is shown in Figure 3.

```
EncryptSection :: = IMPLICIT SEQUENCE {
      encryptType INTEGER,
      encryptData ANY
}
```

Figure 3: ASN.1 Format of Encryption Section

If the section is omitted, then no decryption is required. If the section is present, then the encryptType field contains a number defining the encryption method in use and encryptData contains whatever data, for example a key, which the receiver must have to decrypt the remainder of the message using the type of encryption specified.

Currently no encryption types are assigned.

If the message has been encrypted, data is encrypted starting with the first octet after the encryption section.

## THE REPLY ENCRYPTION SECTION

### Need for Reply Encryption

The reasons for encrypting messages have already been discussed.

The reply encryption section provides the ability for management agents to request that responses be encrypted even though the requests are not encrypted, or that responses be encrypted using a different key or even a different scheme from that used to encrypt the request. A good example is a public key encryption system, where the requesting application needs to pass its public key to the processing agent.

### Format of the Reply Encryption Section

The reply encryption section contains any data required to encrypt the reply message. The ASN.1 format of this section is shown in Figure 4.

```
ReplyEncryptSection :: = IMPLICIT SEQUENCE {
      replyEncryptType INTEGER,
      replyEncryptData ANY
}
```

Figure 4: ASN.1 Format of Reply Encryption Section

If the section is omitted, then the reply should be encrypted in the manner specified by the encryption section. If the section is present, then the replyEncryptType field contains a number defining the encryption method to use and replyEncryptData contains whatever data, for example a key, which the receiver must have to encrypt the reply message.

If the reply encryption section is present, then the reply message must contain an appropriate encryption section, which indicates the encryption method requested in the reply encryption section is in use. The reply message should be encrypted starting with the first octet after the encryption section.

If the reply encryption method requested is not supported by the entity, the entity may not send a reply. It may, at the discretion of the implementor, send a protocol error message. (See below for descriptions of protocol error messages.)

Currently no encryption types are assigned.

## THE AUTHENTICATION SECTION

### Need for Authentication

It is often useful for an application to be able to confirm either that a message is indeed from the entity it claims to have originated at, or that the sender of the message is accredited to make a monitoring request, or both. An example may be useful here. Consider the situation in which an entity sends a event message to a monitoring center which indicates that a trunk link is unstable. Before the monitoring center personnel take actions to re-route traffic around the bad link (or makes a service call to get the link fixed), it would be nice to confirm that the event was indeed sent by the entity, and not by a prankster. Authentication provides this facility by allowing entities to authenticate their event messages.

Another use of the authentication section is to provide access control. Requests demand processing time from the entity. In cases where the entity is a critical node, such as a gateway, we would like to be able to limit requests to authorized applications. We can use the authentication section to provide access control, by only allowing specially authenticated applications to request processing time.

It should also be noted that, in certain cases, the encryption method may also implicitly authenticate a message. In such situations, the authentication section should still be present, but uses a type code which indicates that authentication was provided by the encryption method.

### Format of the Authentication Section

The authentication section contains any data required to allow the receiver to authenticate the message. The ASN.1 format of this section is shown in Figure 5.

```
AuthenticateSection :: = IMPLICIT SEQUENCE {
        authenticateType INTEGER,
        authenticateData ANY
       }
```

Figure 5: ASN.1 Format of Authentication Section

If the section is omitted, then the message is not authenticated. If the section is present, then the authenticateType defines the type of authentication used and the authenticateData contains the authenticating data.

This memo defines two types of authentication, a password scheme and authentication by encryption method. For the password scheme, the AuthenticateSection has the form shown in Figure 6.

```
AuthenticateSection :: = IMPLICIT SEQUENCE {
        authenticateType INTEGER { password(1) },
        authenticateData OCTETSTRING
 }
```

Figure 6: ASN.1 Format of Password Authentication Section

The authenticateType is 1, and the password is an octet string of any length. The system is used to validate requests to an entity. Upon receiving a request, an entity checks the password against an entity specific password which has been assigned to the entity. If the passwords match, the request is accepted for processing. The scheme is a slightly more powerful password scheme than that currently used for monitoring on the Internet.

For authentication by encryption, the AuthenticateSection has the format shown in Figure 7.

```
AuthenticateSection :: = IMPLICIT SEQUENCE {
        authenticateType INTEGER { encryption(2) },
        authenticateData NULL
 }
```

Figure 7: ASN.1 Format of Encryption Authentication Section

This section simply indicates that authentication was implicit in the encryption method. Recipients of such messages should confirm that the encryption method does indeed provide authentication.

No other authentication types are currently defined.

If a message fails authentication, it should be discarded. If the type of authentication used on the message is unknown or the section is omitted, the message may be discarded or processed at the discretion of the implementation. It is recommended that requests with unknown authentication types be logged as potential intrusions, but not processed.

## THE COMMON HEADER

The common header contains generic information about the message such as the protocol version number and the type of request. The ASN.1 format of the common header is shown in Figure 8.

```
CommonHeader ::= IMPLICIT SEQUENCE {
    link IMPLICIT INTEGER,
    messageType IMPLICIT INTEGER,
    messageId IMPLICIT INTEGER,
    resourceId ANY
}
```

Figure 8: ASN.1 Format of Common Header

The link indicates which version of HEMS is in use.

The messageType is a value indicating whether the message is a request (0), reply (1), event (2), protocol error (3) or application error (4) message.

The messageId is a unique bit identifier, which is set in the request message, and echoed in the response. It allows applications to match responses to their corresponding request. Applications should choose messageIds such that a substantial period of time elapses before a messageId is re-used by a particular application (even across machine crashes).

Event messages also use the messageId field to indicate the number of the current event message. By comparing messageId fields from events lost, event values may be detected. The event messageId should be reset to 0 on every reboot, and by convention, the event message with messageId of 0 should always be a "reboot" event. (Facilities should be provided in the event message definition to allow entities which are capable of storing messageIds across reboots to send the highest messageId reached before the reboot.)

The resourceId is defined for ISO compatibility and corresponds to the resource ID used by the Common Management Information Protocol to identify the relevant ISO resource.

DATA SECTION

The data section contains the message specific data. The format of the data section is shown in Figure 9.

```
Data ::= ANY
```

Figure 9: ASN.1 Format of Data Section

The contents of the data section is application specific and, with the exception of protocol error messages, is outside the scope of this memo.

## TRANSPORT PROTOCOL

There has been considerable debate about the proper transport protocol to use under HEMP. Part of the problem is that HEMP is being used for two different types of interactions: request-response exchanges and event messages. Request-response interactions may involve arbitrary amounts of data being sent in both directions, and is believed to require a reliable transport mechanism. Event messages are typically small and need not be reliably delivered.

Public opinion seems to lean towards running HEMP over a transaction protocol (see RFC-955 for a general discussion). Unfortunately, the community is still experimenting with transaction protocols, and many groups would like to be able to implement HEMP now. Accordingly, this memo defines two transport protocols for use with HEMP.

Groups interested in using an implementation of HEMP and the HEMS in the near future should use a combination of the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) under HEMP. TCP should be used for all request-response interactions and UDP should be used to send event messages. Using UDP to support the request-response interactions is strongly discouraged.

More forward looking groups are encouraged to implement HEMP over a transaction protocol, in particular, experiments are planned with the Versatile Message Transaction Protocol (VMTP).

## PROTOCOL ERROR MESSAGES

Protocol error messages are so closely tied to the definition of HEMP that it made sense to define the contents of the data section for protocol error messages in this memo, even though the data section is generally considered application specific.

The data section of all protocol error messages has the same format, which is shown in Figure 10. This format has been chosen to agree with the error message format and ASN.1 type used for language processing errors in RFC-1024, and the error codes have been chosen such that they do not overlap.

```
ProtocolError ::= [APPLICATION 0] implicit sequence {
    protoErrorCode INTEGER,
    protoErrorOffset INTEGER,
    protoErrorDescribed IA5String,
}
```

Figure 10: Data Section For Protocol Error Messages

The protoErrorCode is a number which specifies the particular type of error encountered. The defined codes are:

0 - reserved <not used>

1 - ASN.1 format error. Some error has been encountered in parsing the message. Examples of such an error are an unknown type or a violation of the ASN.1 syntax.

2 - Wrong HEMP version number. The version number in the common header is invalid. Note that this may be an indication of possible network intrusion and should be logged at sites concerned with security.

3 - Authentication error. Authentication has failed. This error code is defined for completeness, but implementations are *strongly* discouraged from using it. Returning authentication failure information may aid intruders in cracking the authentication system. It is recommended taht authentication errors be logged as possible security problems.

4 - ReplyEncryption type not supported. The entity does not support the encryption method requested in the ReplyEncryption section.

5 - Decryption failed. The entity could not decrypt the encrypted message. Note that this means that the entity could not read the CommonHeader to find the messageId for the reply. In this case, the messageId field should be set to 0.

6 - Application Failed. Some application failure made it impossible to process the message.

The protoErrorOffset is the number of the octet in which the error was discovered. The first octet in the message is octet number 0.

The protoErrorDescribed field is a string which describes the particular error. This description is expected to give a more detailed description of the particular error encountered.

## APPENDIX OF TYPES

This section lists all ASN.1 types defined in this document.

HEMP Types

```
HempMessage ::= [0] IMPLICIT SEQUENCE {
    [0] IMPLICIT EncryptSection OPTIONAL,
    [1] IMPLICIT ReplyEncryptSection OPTIONAL,
    [2] IMPLICIT AuthenticateSection OPTIONAL,
    [3] IMPLICIT CommonHeader,
    [4] IMPLICIT Data }

EncryptSection :: = IMPLICIT SEQUENCE {
    encryptType INTEGER,
    encryptData ANY
}

ReplyEncryptSection :: = IMPLICIT SEQUENCE {
    replyEncryptType INTEGER,
    replyEncryptData ANY
}

AuthenticateSection :: = IMPLICIT SEQUENCE {
    authenticateType INTEGER,
    authenticateData ANY
}

CommonHeader ::= IMPLICIT SEQUENCE {
    link IMPLICIT INTEGER,
    messageType IMPLICIT INTEGER {
        request(0), reply(1), event(2),
        protocol error (3), application error(4)
    }
    messageId IMPLICIT INTEGER,
    resourceId ANY
}

Data ::= ANY
```

Protocol Error Types

```
ProtocolError ::= [APPLICATION 0] implicit sequence {
    protoErrorCode INTEGER,
    protoErrorOffset INTEGER,
    protoErrorDescribed OCTETSTRING
}
```

REFERENCES

ISO Standard ASN.1 (Abstract Syntax Notation 1). It comes in two parts:

International Standard 8824 -- Specification (meaning, notation)
International Standard 8825 -- Encoding Rules (representation)

The current VMTP specification is available from David Cheriton of Stanford University.